CHAPTER

EIGHTEEN

DISTRIBUTED PROBLEM SOLVING

So far in this book we have described several models and languages for understanding and controlling distribution and presented some (relatively) mature and specific algorithms for distributed database systems. In this section we become highly speculative—describing particular algorithms for organizing nonexistent machines, often to do tasks that we do not know how to do on conventional, sequential machines.

After all, why would anyone want a distributed computing system? Some tasks, such as databases and sensing (which we discuss below), are naturally distributed — they reflect the distribution of their input sources and output destinations. However (to echo one of our earlier themes), the real promise of the megacomputer lies in its potential for vast increases in processing. What tasks require so many cycles? Processing is important for problems involving search, where we use "search" in its broadest sense. That is, an intelligent system *searches* in the space of possible actions for the solution to its particular problem.

Of course, assignments in an introductory programming course can also involve search. In general, search tasks span a range from well-structured domains to ill-structured domains. In a well-structured domain, the computer merely sifts through a mass of data to find an answer. At any point it is clear which operation is to be applied to which data items. The control structure of the search can be predicted before the program is run. Most primitive operations fail and

this failure is forgotten. That is, the choice of what to consider next is usually independent of most of what has already been discovered.

In an ill-structured domain there are many potentially useful actions at any point. Selecting the most appropriate direction to follow is itself a problem requiring solution. Distributed computing is most intriguing for this class of problems because of the possibility of *really* pursuing many different paths at the same time, extending those that appear promising while deferring those that seem barren. In effect, this scheme allows the real resource consumption of subgoals to influence the processing agenda.

Of course, distributed systems can be used for well-structured search domains. Such domains allow the task to be decomposed into processor-size pieces before it is run. The processors run in parallel, with only a minimum of communication required to integrate their results.

Ill-structured domains do not provide the luxury of predefined decomposition. Instead, the course of problem solving shapes the future direction of the program. Such ill-structured domains often have multiple knowledge sources, each of which can contribute to the solution of some problems. These sources need to communicate queries, hypotheses, and partial results—to *cooperate* in problem solving.

Functionally Accurate Cooperative Systems

Victor Lesser and Daniel Corkill [Lesser 81] characterize contemporary distributed systems as being "completely accurate, nearly autonomous" systems. These systems are completely accurate because they operate on complete and correct information. They are nearly autonomous because each process has an almost complete database of the information it needs. In completely accurate, nearly autonomous systems, the interprocess relationship is usually that of a calling program and called subroutine (or, more figuratively, *master-slave*).

Lesser and Corkill argue that the best way to solve some problems is by decomposing them into local subproblems that operate on incomplete data. This division seems natural for dealing with problems that use many diverse knowledge sources (such as speech understanding in the Hearsay model [Erman 80]) and for problems involving noisy, distributed data (such as distributed sensing).* They suggest that these systems should be organized as "functionally accurate, cooperative" distributed systems. In such systems, the processing nodes use incomplete data and exchange partial results. Functionally accurate systems produce acceptable, though not perfect, answers. Such systems must deal with uncertainty inherent in both the problem and any intermediate results. Functional accuracy can be achieved by processes

* In a *distributed sensing system*, sensing/processing units focus on particular parts of space and cooperate in tracking objects through the whole of the space. For example, a distributed radar system that tracks incoming missiles is a distributed sensing system.

Figure 18-1 A hierarchical control structure.

that act as *iterative coroutines*, communicating partial results and extending the partial results of other processes. These systems are therefore inherently cooperative.

Cooperation structures How can systems be structured to focus control and provide direction? One obvious mechanism is hierarchy—some topmost node (the general) divides the original task into subproblems, and then distributes these subproblems to the second level of command (the colonels). The colonels repeat this subdivision through the ranks until "private-size" tasks reach the nodes that do the primitive work. Figure 18-1 shows a hierarchical control structure. The results of processing progress from the leaves of the hierarchy to the root. At each level, an officer node synthesizes the results of its immediate subordinates. This synthesis may require additional requests of the lower-level processing nodes. Thus, this organization leads naturally to communication up and down the hierarchy, but not across subtrees.

Hierarchical organization has the advantage that there is *some* control path for focusing the system if the problem environment changes radically. However, this organization may prove unwieldy in a distributed system whose elements themselves possess considerable processing ability. The higher-level officer nodes may be a communication bottleneck; the problem may not decompose into nearly-independent subtasks, and the system as a whole is delicately dependent on the health (nonfailure) of the top-level command.

Several alternative *heterarchical* control organizations for distributed systems have been proposed. In this chapter we consider three of these, Distributed Hearsay, Contract Nets, and the Scientific Community Metaphor.

18-1 DISTRIBUTED HEARSAY

The Hearsay-II system [Erman 80] is a speech understanding system.* It synthesizes the partial interpretations of several diverse knowledge sources into a coherent understanding of a spoken sentence. These *knowledge sources* cover domains such as acoustics, phonetics, syntax, and semantics. Each knowledge source iterates through the process of first hypothesizing a possible interpretation of some part of the current data and then testing the plausibility of that hypothesis. The various knowledge sources form new hypotheses by using both a priori knowledge about the problem domain (the information about speech understanding incorporated in the knowledge source when it was created) and those hypotheses already generated in the problem-solving process. Because knowledge sources work with imperfect a priori knowledge and noisy input signals, many of the hypotheses they create are incorrect. Of course, conclusions that other knowledge sources draw from incorrect hypotheses are also suspect. To avoid focusing the system on a single, inappropriate solution path, each knowledge source can generate several possible interpretations of input data, associating a credibility rating with each.

Knowledge sources communicate by reading and writing a global database called the *blackboard*. The blackboard has several distinct levels. Each level holds a different representation of the problem space (Figure 18-2). Typical blackboard levels for speech understanding are sound segments, syllables, words, and phrases. The knowledge sources are pattern-action productions; if the information on the blackboard matches the pattern of a knowledge source then its action can be executed. This action usually writes new hypotheses on the blackboard. Most knowledge source patterns refer to only a few contiguous blackboard levels.

At any time, many knowledge sources are likely to have patterns that match the contents of the blackboard. The *scheduler* decides which knowledge source is to be executed next, choosing the knowledge source whose action has the highest priority. An action's *priority* is an estimate of the impact (reduction in problem uncertainty) of the information generated by executing its pattern. Hearsay-II also has a *focus-of-control* database that contains meta-information about the system's state. This information is used both to estimate the impact of actions and to redirect a stagnating system.

Since many actions are potentially executable at any time, Hearsay-II would seem to be a naturally concurrent system. This proves to be the case. Experiments have shown that a shared-memory, multiprocessor Hearsay-II implemen-

* The Hearsay-II architecture has been used for other knowledge-based interpretation tasks, such as sonar interpretation [Nii 78] and image understanding [Hanson 78]. To be accurate, we should distinguish between the architectural organization (knowledge sources, blackboard, etc.) of Hearsay-II and the particular application of Hearsay-II to speech understanding. However, for the sake of simplicity we merge the two, speaking of Hearsay-II as a speech understanding system. We rely on the reader to generalize the concepts to the Hearsay-II architecture.

Figure 18-2 Knowledge sources and the levels of the blackboard.

tation runs significantly faster than the original system [Fennell 77]. However, because of its reliance on a centralized, global blackboard, Hearsay-II is not trivial to distribute.

Erman and Lesser's approach to distributing Hearsay-II makes each node of the system a full Hearsay-II system in its own right, complete with knowledge sources, blackboard, scheduler, and focus-of-control database [Lesser 80]. This is possible because speech understanding naturally divides along two dimensions, blackboard level and time. The blackboard has multiple levels, and few knowledge sources mention information on more than two contiguous levels. The speech signal is itself distributed in time—typically, signals a few seconds apart interact only at the highest semantic levels. Thus, by overlapping the signal covered by different nodes, Distributed Hearsay-II systems need to communicate only semantic information. This distribution of tasks by the two-dimensional problem space produces a "near neighbor" communication pattern. The decision of which information to communicate is also entrusted to knowledge sources—particular *transmit* actions are matched by appropriate *receive* patterns.

Erman and Lesser's experiments with a Distributed Hearsay-II system divided the knowledge-level/time space only along the temporal dimension. Each logical node was statically assigned a continuous segment of speech signal, with the segments of neighboring nodes overlapping. Thus, each node had a full set of knowledge sources. Figure 18-3 shows the task division among the nodes. Distributing Hearsay-II prompted a few modifications to the speech understanding knowledge sources. Erman and Lesser added communication knowledge sources, modified those knowledge sources that depended on distant information, and corrected those knowledge sources that contained implicit assumptions about the sequential nature of the overall processing. They discovered that a (simulated) three-node Distributed Hearsay-II system proved accurate on a set of sample utterances and produced a slight (10 percent) improvement in overall system

speed. The system was also somewhat immune to communication errors. Even when many internode communications were discarded, it still understood most sample sentences. Distributed Hearsay-II was thus self-correcting and functionally accurate. This result is not surprising; the original Hearsay-II architecture was designed to deal with noisy and incomplete data. This resilience carried over into the distributed environment.

The Hearsay-II speech understanding system was easy to distribute because the problem domain and knowledge sources themselves have a natural distribution. Few knowledge sources reference noncontiguous levels of the blackboard; the understanding of a particular segment of speech is related only semantically to another segment seconds later. Our next two approaches, Contract Nets and the Scientific Community Metaphor, consider the organization of interacting knowledge sources without respect to the geometry of the underlying problem structure.

Figure 18-3 Task division in Distributed Hearsay-II.

18-2 CONTRACT NETS

Reid Smith and Randall Davis propose the Contract Net Protocol as an architecture for organizing distributed systems, particularly for the problem of distributing tasks among a set of (potentially heterogeneous) nodes [Smith 78, Smith 81a]. They start with the idea of expert problem solvers who need to communicate tasks and solutions. Smith and Davis argue that such a collection of experts need *problem-solving protocols*, much as computer networks need communication protocols. The Contract Net Protocol draws its inspiration from the activities surrounding the negotiation of commercial contracts.

A *Contract Net* is a set of autonomous processing nodes that communicate according to the rules of the Contract Net Protocol. Individual subtasks of the global problem are called *contracts*. A node that needs a subtask performed advertises (broadcasts) the existence of that task to the other nodes of the network. Those nodes that have the resources or expertise to solve the task *bid* on it, returning the bids to the broadcasting node (the *manager* of the task). On the basis of the information in the bid responses, the manager node *awards* the contract to one of the bidders, the *contractor*. The contractor can then break the task of the contract into further subtasks, letting out contracts on those subtasks.

Unlike hierarchical structures, the Contract Net Protocol calls for negotiation between nodes. Nodes evaluate announcements, bidding only on those of interest. Managers evaluate bids to select the most appropriate bidder. The protocol thus ensures a mutual selection.

A manager with a contract can broadcast the terms of that contract to all other nodes. However, broadcasting wastes precious communication bandwidth. A manager that knows which nodes are potential bidders sends a *limited broadcast* to only those nodes. A manager that can narrow the potential contract to a single node sends just that node a *point-to-point announcement*. The Protocol also allows *directed contracts*, awarded by a manager to a specific node without bidding, and *request-response sequences*, used to obtain immediate information without negotiation.

Task announcements contain three kinds of information. The *task abstraction* is a brief description of the task. The *eligibility specification* lists criteria that potential bidders must meet. And the *bid specification* details the form of desired bids. The bid specification resembles an application form with blanks to be filled in; potential contractors need to return only values for the blanks, instead of retransmitting the entire form.

After a manager has received bids, it selects a successful bidder and awards the contract to that bidder. This award contains a *task description*, a complete specification of the task. On completing the task, the contractor returns a *task report* to the manager, which includes the results of performing the task. Managers may also terminate unfinished contracts still in progress.

Smith and Davis cite distributed sensing as a possible application of Contract Nets. Their hypothetical sensing system has many nodes that communicate over

a common broadcast channel. The nodes themselves have sensing or processing capabilities (or both). A *sensing node* detects traffic in its neighborhood and performs low-level feature analysis. A *processing node* integrates and processes the data from sensing nodes. A processing node does not have to be physically near a sensing node to use its results. A single *monitor node* is responsible for managing the entire task and communicating with the outside world. The other nodes are at fixed positions. These nodes know their positions, but this knowledge is not known a priori by the monitor.

Just as the knowledge in speech understanding naturally divides into different levels, distributed sensing has distinct levels: signal processing, signal grouping, vehicle detection, area mapping, and global mapping. Related signals from a single sensor are formed into signal groups; several signal groups detected from different locations compose into a vehicle. Once discovered, vehicles are subject to analysis for type, location, and projected course. From the vehicles in a region the system develops an understanding of the traffic in that region; from regional understanding comes a global understanding. This natural hierarchy leads to a processing hierarchy. Figure 18-4 shows one such organization of nodes.

The first task of the system is to divide the traffic space into areas. The monitor node (knowing the names of potential area processors) broadcasts a task announcement to them, requesting bids on area division. The area processors respond with bids; each bid includes the bidder's location. From the replies, the monitor partitions the space, awarding area contracts to some of the bidders, the area managers. These managers continue this process recursively, seeking signal-group processing nodes. However, since the area nodes do not know which signal-group nodes are in their areas, they must broadcast their task announcements. These announcements require (as an eligibility specification) that the signal-group nodes must be within the area node's area. The process continues through to signals, thus defining the initial state of the sensing tree. Nodes that do not already have tasks (or have the capacity to accommodate additional tasks) continue to listen for task announcements and bid for contracts.

The same organization that is used to create the sensing hierarchy is used during system operation. A signal sensor that detects a signal reports to its group manager. This group manager then integrates this signal with its existing signal group or attempts to form a new signal group. The group manager reports new groups to the area contractor. Using the contract announcement and bidding procedure, the area contractor either finds the vehicle contractor that is already monitoring that signal or creates a new vehicle monitor to do that monitoring. If the vehicle monitoring task requires help with localization or course prediction, the vehicle monitor issues subcontracts.

We introduced distributed problem solving by arguing that in ill-structured, distributed search, dividing the search space is itself a major task. The Contract Net formalism takes this idea to heart, providing mechanisms for distributing tasks. Just as programming languages do not prescribe per se which programs are to be written in them, the Contract Net Protocol does not specify what

Figure 18-4 The distributed sensing hierarchy.

contracts should be let and bid or how bid specifications should be phrased. Contract Nets provides only the framework for internode negotiations on the problem-solving issues.

18-3 THE SCIENTIFIC COMMUNITY METAPHOR

Paradigms draw their inspiration from analogy—the transfer of the key features of one domain to another. The Hearsay-II architecture is guided by analogy to

people who communicate on a shared blackboard; Contract Nets, by analogy to businesses negotiating for subcontracts.

Perhaps the ultimate human refinement of the problem-solving process is the activity of science itself. William Kornfeld and Carl Hewitt have investigated the idea of treating machine problem solving analogically to the problem-solving structure of an idealized scientific community [Kornfeld 81]. Scientists, far more than the knowledge sources of Hearsay-II, incorporate active problem solving. And like the societies of which they are a part, scientific communities use the economics of funding as an essential control mechanism.

Kornfeld and Hewitt identify several aspects of the organization of scientific research that they feel are important to mimic in distributed problem-solving systems: monotonicity, commutativity, parallelism, and pluralism. *Monotonicity* refers to the monotonic increase in the store of scientific knowledge—early results may be later contradicted, but their vestiges remain. For scientific communities, archived journal volumes embody this monotonicity. *Commutativity* is the ability of scientists to draw both on work already completed and on work still to be done. That is, it does not matter which came first, the "answer" or the "question"—the two can be matched in either case. *Parallelism* results from scientists working concurrently, using this concurrence to guide resource allocation and search direction. And *pluralism* refers to the ability of a scientific community to entertain multiple hypotheses at any time, with no hypothesis ever achieving the status of "absolute truth."

Kornfeld and Hewitt have developed the language Ether to express highly parallel, "scientific community" algorithms. Ether is based on the message-passing theme of Actors (Chapter 11) and extends the demon ideas of artificial intelligence languages such as Planner [Hewitt 69] and Micro-Planner [Sussman 70]. These languages revolve around a global database. Each demon has a pattern and an action. The demon "watches" the database, and when the information in the database matches its pattern, it executes its action.*

In Planner, demons recognized only those database changes that happened *after* their creation. Ether proposes a new kind of demon, the sprite. Like Planner demons, sprites recognize database entries that match their patterns. However, unlike demons, sprites are commutative. A sprite that matches a data item matches that item, regardless of whether the sprite was created before or after the data item. Ether systems thus resemble the interaction of scientists and scientific libraries. A researcher interested in a given topic finds papers on that topic regardless of whether the papers were written before or after her interest was aroused. Similarly, a sprite interested in a particular fact in the database finds that fact regardless of when it was created.

* Resemblance to the knowledge sources and blackboard of Hearsay-II is not coincidental. Demons have been a recurring A.I. theme with more different instantiations than we care to list. However, Planner was certainly one of the first systems to explicitly support them. *Production systems* are the generalization of pattern/action systems. A good overview of the use of production systems in A.I., circa 1976, is Davis and King [Davis 76].

Since many sprites can be active at any time, Ether is a parallel system. And Ether supports a form of state vector or possible world for dealing with multiple, competing hypotheses, thus providing a mechanism to support pluralism.

Combinatorial implosion Kornfeld presents the example of finding the *covering set* of a predicate using sprites [Kornfeld 82]. We are given a set of propositional predicates, $P = \{p_1, p_2, \ \ldots \ , p_k\}$ and a predicate $\mathcal{P}$ such that

$$\mathcal{P} \supset p_1 \vee p_2 \vee \cdots \vee p_k$$

The problem task is to determine all subsets S of P such that:

$$\mathcal{P} \supset S$$

and

$$\text{there is no proper subset } R \text{ of } S \text{ such that } \mathcal{P} \supset R$$

That is, we want all the "minimal covering subsets" of P. For example, we imagine our propositions ranging over the base predicates A, B, C, D, and E. If our original set $\{p_1, p_2, p_3, p_4, p_5\}$ is

$$\begin{array}{lcl} p_1 & = & A \vee B \\ p_2 & = & C \vee D \\ p_3 & = & A \vee C \\ p_4 & = & B \vee D \\ p_5 & = & E \end{array}$$

and our given predicate $\mathcal{P}$ is

$$\mathcal{P} = A \vee B \vee C \vee D \vee E$$

then the minimal subsets, S, are

$$\{p_1, p_2, p_5\}$$

and

$$\{p_3, p_4, p_5\}$$

We say that a set S for which $\mathcal{P} \supset S$ is a *working set*, while a working set with no working, proper subsets is a *minimal set.*

This problem is amenable to both top-down and bottom-up solutions. In the top-down solution, one starts with the entire set, P. We consider all subsets of P formed by removing one element from P. If none of those sets works, then P is a minimal set (and should be added to the set of answers). If any one of those subsets works, then we need to apply the process recursively to each working subset.

This algorithm is easily expressed in Lisp. We represent sets of propositions as lists; the set $\{p_1, p_2, p_4\}$ becomes the list (p_1 p_2 p_4). We imagine having the following auxiliary functions and constant:

(working s) ≡ is true if s works with respect to the global predicate $\mathcal{P}$.
(remove x l) ≡ is a list of all the elements of l except x.
(result w) ≡ adds w to the list of answers if it is not already there.
(mapcar f l) ≡ applies function f to each element of l, returning a list of the results. Mapcar is a standard Lisp function.
(mapc f l) ≡ applies function f to each element of l, returning nil. Mapc is a standard Lisp function.
(mapconc f l) ≡ applies function f to each element of l. Each application should yield a list of values. These lists are appended together to form the function's result. (Actually, the lists are destructively joined together, an irrelevant detail for our purposes.) Mapconc is a standard Lisp function.
P ≡ the list that represents the entire set of propositions.

The top-down function is as follows:*

```
(top_down l) ≡
    (top_down_recur l
                    (keep_working (mapcar (lambda (x) (remove x l))
                                          l)))

(top_down_recur l w) ≡
    (cond ((null w) (result l))
          (t (mapc top_down w)))

(keep_working m) ≡
    (mapconc (lambda (s)
                 (cond ((working s) (list s))
                       (t nil)))
             m)
```

The program is run as (top_down P). Before running this program the answer list should be set to nil.

In the bottom-up solution, we successively create all subsets of P, from the empty set to the entirety of P. At each stage, we add each set that works and is not a superset of any minimal set to the collection of minimal sets. For this program, we need the additional auxiliary functions

(superset_working l) ≡ is true if l is a superset of any already-found answer.

* We have written this program with two auxiliary functions to avoid deluging the reader with embedded lambda expressions.

(member x l) ≡ is true if x is a (top-level) element of the list l. Member is a standard Lisp function.

The bottom-up program is as follows:

```
(bottom_up l) ≡
    (cond ((null l) nil)
          (t (mapc bottom_up_one l)
             (bottom_up (successors l))))

(bottom_up_one s) ≡
    (cond ((superset_working s) nil)
          ((working s) (result s)))

(successors l) ≡
    (mapconc
        (lambda (m)
            (mapconc
                (lambda (x)
                    (cond ((member x m) nil)
                          (t (list (cons x m)))))
                P))
        l)
```

-- *The embedded mapping functions are the Lisp equivalent to Pascal embedded for loops.*

This program is run as (bottom_up (quote (nil))).

Which program, top_down or bottom_up, is better? The top-down solution is faster if the minimal subsets are large with respect to the original set; the bottom-up solution is faster if they are small.* Each algorithm is already amenable to some immediate concurrent acceleration, because the mapping functions (mapc, mapcar, and mapconc) can apply their functional arguments to the elements of their list arguments in parallel.

Kornfeld observes that a large improvement in execution time can result from running both algorithms concurrently if each passes information about its discoveries to the other. More specifically, when top_down finds that a set does not work, then no subset of that set works; when bottom_up finds that a set works, then no superset of that set is minimal. Many algorithms

* These programs are also wasteful, in that many subsets of P are generated repeatedly and some answers are found several times. Exercise 18-5 asks for a modification of the top-down algorithm to avoid generating redundant subsets.

are of exponential complexity — the number of elements that need examining "explodes" (increases exponentially) as the size of the problem increases. The top-down and bottom-up algorithms have this property. If P is a set of size k, then bottom_up considers 2^k different possible sets in its search for minimal sets. However, the combination of several search algorithms, operating cooperatively and in parallel, can eliminate this combinatorial explosion, producing instead a combinatorial implosion. Kornfeld proposes the Ether language as an appropriate vehicle for describing combinatorially implosive algorithms.

Ether is based on sprites. The syntax

$$\text{(when <trigger> <command}_1\text{> } \ldots \text{ <command}_k\text{>)}$$

defines the action of a sprite that, when it recognizes database entries that match <trigger>, executes actions <command$_1$> ... <command$_k$>. Function assert enters items into the database. To set a sprite working, it must be activated.

The sprite that recognizes working sets and asserts the nonminimality of their supersets is

```
(not_minimal_upwards) ≡
    (when ⟨Works S⟩
        (foreach q ∈ P
            (if (not (q ∈ S))
                (assert ⟨Works ({q} ∪ S)⟩)))))
```

Similarly, the sprite that asserts that subsets of nonworking sets are nonworking is

```
(not_working_downwards) ≡
    (when ⟨NotWorks S⟩
        (foreach q ∈ S
            (assert ⟨NotWorks (S − {q})⟩))))
```

Of course, the value of working simultaneously from both ends is that redundant searching can be eliminated. Ether provides such a capability with processes that can be explicitly destroyed. Ether calls processes *activities*. Function NewActivity creates a new activity; function Execute starts an activity running a particular piece of code. A *sprite* is an activity that runs a pattern/action program. Executing stifle on an activity aborts it.

In our example, evaluation of (working S) asserts (in the database) either ⟨Works S⟩ or ⟨NotWorks S⟩. Sprites whose pattern refers to one of these assertions will then be able to execute their actions.

```
(top_down S) ≡
    (foreach q ∈ S
```

```
        (let ((activity (NewActivity)))                    -- Create a new sprite
            (Execute (working (S − {q})) activity)            and call it "activity."
            (when ⟨Works (S − {q})⟩
                  (stifle activity)
                  (top_down (S − {q})))
            (when ⟨NotWorks (S − {q})⟩
                  (stifle activity))))
    (when (∀ q ∈ S ⟨NotWorks (S − {q})⟩)                    -- This sprite has a
        (assert ⟨Minimal S⟩)))                                quantified pattern.
```

Similarly, function bottom_up is

```
(bottom_up S) ≡
    (foreach q ∈ P
        (if (not (q ∈ S))
            (let ((activity (NewActivity)))
                (Execute (working (S ∪ {q})) activity)
                (when ⟨NotWorks (S ∪ {q})⟩
                      (stifle activity)
                      (bottom_up (S ∪ {q})))
                (when ⟨Works (S ∪ {q})⟩
                      (stifle activity)
                      (assert ⟨Minimal (S ∪ {q})⟩))))))
```

The entire program is run as

```
(progn
    (Execute (top_down P) (NewActivity))
    (Execute (bottom_up { }) (NewActivity))
    (Execute (not_working_downwards) (NewActivity))
    (Execute (not_minimal_upwards) (NewActivity)))
```

which sets four initial sprites running to solve the task. The last two sprites in this list are accelerators facilitating the passing of results through the system. When the activity created by this function has quiesced, the database contains Minimal assertions for exactly the minimal sets.

This example uses a simple structure (database pairs such as ⟨Works x⟩) to encode the information discovered by the sprites. Hewitt, Kornfeld, and de Jong [Kornfeld 81; Hewitt 84] argue that distributed systems based on communication need to be more complex; they must incorporate distinguishable world viewpoints, descriptions of system objects, sponsorship-based control, and elements of self-knowledge and self-reference. They call such systems *Open Systems*.

Viewpoints support the relativization of beliefs. A system with *viewpoints* allows the creation of possible worlds, the assertion of different hypotheses in

different worlds, and the deduction of varying conclusions based on these differing hypotheses. A major issue in the construction of such an architecture is the inheritance of properties between subworlds.*

The idea of descriptions is that the description of what something is should be separated from the details of its implemention. This echoes both the abstract data type theme of separating abstract specification from implementation (Section 2-2) and the ancient A.I. debate over procedural and declarative representations [Winograd 75].

A major factor controlling the direction of scientific research is the allocation of research funds by sponsoring organizations. More promising research is more likely to be funded. In Ether, this idea is reflected in the requirement that all sprite triggering is to be performed under the control of a sponsor that is working on a particular goal. An explicit goal function associates goals with sponsors. The stifle command of the covering sets program is also an example of sponsor-based control.

Finally, since distributed problem solving involves many elements of negotiation and control, distributed problem-solving systems need to have some elements of self-knowledge and self-reference. When a subsystem is floundering or reaching contradictions, it needs mechanisms to discover and analyze the problem.

In summary, Hewitt and his coworkers propose extending the communication basis of Actor systems (Chapter 11) to distributed problem-solving systems. Their investigation has identified many fundamental aspects of such systems. These include the need to base distributed systems on communication and to model them on sophisticated problem-solving mechanisms.

18-4 SYNTHESIS

Different programming languages have different pragmatic characteristics. Individual languages lend themselves to certain tasks and suggest particular algorithms for those tasks. We find similar specializations in the pragmatics of distributed problem-solving architectures. Distributed Hearsay takes advantage of the regular geometry of certain problem spaces. It divides a task into chunks that overlap on that space and allocates one chunk to each process. An interesting open question is whether the Distributed Hearsay-II architecture can profitably be applied in domains that lack an appropriate geometry. The Contract Net Protocol focuses on the task distribution aspect of problem solving, taking seriously the idea that task distribution is a problem that requires solution as much as any other. The Protocol suggests modeling task distribution on

* Possible world semantics is yet another subject with a vast literature, spanning both philosophy and artificial intelligence. Some of the more interesting A.I. ideas are those of Moore [Moore 79] and Weyhrauch [Weyhrauch 80].

a simplified contract economy. The Scientific Community Metaphor argues that the ultimate distributed systems will need to be communication-based reasoning systems. Such systems will require richer representations of data and more complex inference patterns than simple serial systems have so far achieved.

Of course, each of these formalisms has its limitations. Distributed Hearsay-II is still limited by the geometry of the problem space. Contract Nets is a general but low-level tool for organizing systems. It recognizes that the elements of the problem-solving system will not even know *where* to find the expertise they need. Its resolution is to broadcast requests for subtask solution. However, depending on the underlying architecture, broadcasting can be expensive. It may be a mistake to encourage broadcasting at so low a system level. Contract Nets also requires that the solutions of tasks be funneled back to the originator of that task. This precludes continuation-based architectures.

The Scientific Community Metaphor is probably the most advanced in recognizing the attributes required of a coordinated computing system. But that advanced perspective is a major obstacle to using these ideas in the near future. The Metaphor demands knowledge representation and reasoning beyond the capabilities of current systems. Journal articles may be permanent, but typical articles mention a vast volume of bibliographic and documentary evidence in the process of drawing only narrow conclusions. Knowledge representation systems have only begun to deal with issues involved in dependencies and truth maintenance. Adding the complexities of communication complicates matters further. The Metaphor's introduction of sponsors acknowledges that distributed problem-solving systems will need to devote much of their resources to self-monitoring.

Programming languages have evolved in the programming environment, improving on the (perceived) inadequacies of earlier languages and propelled by developments in associated mathematical theories such as formal languages and semantics. We expect that distributed problem-solving architectures will evolve in the same way.

PROBLEMS

† **18-1** Lesser and Corkill, in a demonstration of the depth of cooperation needed for serious decentralized control, propose the following problem [Lesser 78, p. 8]:

> Consider a demand bus system where a fleet of buses is to serve an urban area. Upon arrival at a bus stop, a customer might dial his desired destination on a selector device and this information would be used to plan bus routing dynamically. There are a number of elements which must be considered in such a system: buses should be kept reasonably full but not overloaded; the total mileage of the fleet should be kept as low as possible; the mean service time (waiting time at the bus stop and riding time) should be kept small; the maximum service time of any one customer should not exceed a reasonable amount (a customer waiting to get to Fifth and Main should not have to ride around all day merely because no one else needs to go near that location); and the system should be able to monitor and respond to special events (e.g., different traffic patterns at different times of

the day, concerts, athletic events, local weather conditions, bus breakdowns, stalled traffic, etc.)

Lesser and Corkill propose that any solution involve a limited broadcast transmitter/microprocessor on each bus and at each bus stop. They argue that such processing elements must not only retain a local view, but must also achieve distributed control to respond to more global conditions.

Using the language or model of your choice, program a solution to the distributed demand-driven bus system problem.

† **18-2** Lesser and Corkill also propose the Distributed Processing Game [Lesser 78] as a vehicle for exploring distributed control. The game they describe is quite general; the rules are parameterized differently for each play. For the sake of simplicity, we present only a single instantiation of the Distributed Processing Game. Also, for simplicity and tractability of implementation, we have modified some of the rules.

The game is a two-team game, played on a finite section of a plane 2000 by 2000 units large [that is, all points (x, y) such that $-1000 \leq x, y \leq 1000$]. Each team has 24 mobile nodes (the rovers) and a single, stationary home node. At the start of the game, random locations are selected for the home nodes and the players distribute their rovers within a 20-unit radius of their home nodes. Each rover has a unique identity (a number from 1 through 24) and knows its own initial location and the location of its home. The object of the game is to destroy the opponent's home node before one's own home is destroyed.

The rovers are equipped with sensors, communication devices, and energy weapons. The game proceeds in discrete steps, which are alternating team's turns. A turn consists of:

(a) Each rover moves to any spot within 10 units of its current location.

(b) Each node senses its environment. It becomes aware of the location and identity of any node within 20 units. It also becomes aware of the number of friendly and the number of unfriendly nodes within 40 units. It does not find out the identity or specific location of any unit more than 20 units away.

(c) Each node broadcasts a communication.

(d) Each node receives the communications of all nodes on its team within 100 units.

(e) Each rover can point its weapon in any direction, focus it to an angle θ, $15° \leq \theta \leq 90°$, and shoot. The weapon covers an area of 36π square units; the smaller the angle θ, the longer the covered range. Any rover that is in the coverage of the firing weapons of two or more rovers is destroyed (removed from the game). (This also includes rovers of the attacking team.) A "last words" message can be left by a rover just before it is annihilated. This message can include the identity and location of its attackers. It is received on its team's next turn. Any rover that is hit by only a single shot is unscathed. Such a rover becomes aware of the location of its attacker.

On the other hand, home nodes accumulate damage. Each rover that attacks a home node adds one damage unit to it. The first home node to accumulate 20 damage units is destroyed (resulting in a loss for that team). Figure 18-5 shows a pair of attacks.

After the completion of one side's turn, control passes to the other team for its five-step turn. Since attacking requires cooperation, this game is a good test of cooperation strategies.

These rules designate a large amount of computation. But it's just as well, because the various nodes are all processes. In particular, each (human) player *programs* her team's nodes. The game is then run free of human intervention.

The advanced version of the game introduces errors into communication and sensing. That is, in the advanced game, there is a finite probability (say, 15 percent) that any sensing or communication message is lost.

Figure 18-5 Attack sectors.

We have selected the above numbers arbitrarily. Clearly, the game can be varied in other ways. For example, instead of alternating, teams could move simultaneously; messages between nodes could be restricted to some particular size; and so forth.

18-3 Compare and contrast demons and guarded statements.

18-4 Modify the Lisp program for the top-down covering set problem so that it deals abstractly with sets, not lists. Invent any set-primitive and set-mapping functions you may need, such as union, subset, and mapset.

18-5 Modify the programs for the top-down covering set problem so that top_down is never called on any set more than once. (Hint: Provide an ordering on the elements of the original set. Associate with each recursive subset an element "beyond which that subset does not generate new sets.")

18-6 Does running the top_down and bottom_up Ether programs concurrently solve the combinatorial explosion problem in all cases?

† **18-7** By the experimental or analytic method of your choice, determine the expected degree of improvement gained by running the two algorithms concurrently.

18-8 What parallels can you draw between the monotonicity of Ether systems and the monotonicity of the increasing fixedness of frons lists (Chapter 12)?

† **18-9** Societies develop economic systems to organize, control and distribute the results of economic labor. From the time of Adam Smith to the present day, economic systems have been proposed and analyzed. Three of the most interesting systems (from the point of view of coordinated computing) are centralized, planned economies (the current Soviet model) where a centralized authority distributes all tasks and resources; *laissez faire*, free market economies (the "ideal" of the American economy) where the marketplace is the only control; and mixed planned/decentralized economies (the French and Japanese economies) where the government sets goals and provides incentives to meet them. Clearly, centralized planning is the "subroutine structure" of conventional programming. Contract Nets captures some aspects of economies with the idea of contracts; sponsors in Ether capture another. A *laissez faire* economic model

would merge the two with a real sense of currency. That is, Contract Nets gives us processes that compete for work but have no drive for accumulation (why are all these tasks bidding?); sponsors give us funding agencies but no funds (and no funding action except project cancellation). How would a coordinated computing system organized as a free market be structured? What elements of centralized planning could be introduced into such a system to improve its performance? Consider the ability of a central controller to shift system organization through taxation policy.

† **18-10** Mammalian neural systems (particularly the human neural system) have proved to be ideal distributed problem solvers, combining with great skill distributed sensing, intelligent processing, and effective manipulation. The neural system combines a high degree of redundancy with regularly-patterned control paths. It makes significant use of inhibition to prevent undesired processing. The nervous system is also amazingly complex. Read Kent's book, *The Brains of Men and Machines* [Kent 81], and design a coordinated computing system based on neural principles.

† **18-11** Individual businesses achieve their goals though composed of many independent processing elements (the workers). Mark Fox suggests that business organizations can serve as a model of system design. He develops a design language that incorporates facets of the business organization metaphor [Fox 79; Fox 81]. Design a coordinated computing system based on business and management paradigms. Another possible starting place for working on this problem is Galbraith's *Organizational Design* [Galbraith 77]. That book is a good introduction to management concepts used in business organizations.

REFERENCES

[**Corkill 83**] Corkill, D. D., and V. R. Lesser, "The Use of Meta-level Control for Coordination in a Distributed Problem Solving Network," *Proc. 8th Int. J. Conf. Artif. Intell.*, Karlsruhe, Germany (August 1983), pp. 748–755. In this paper, Corkill and Lesser extend their work on Distributed Hearsay-II, using the problems of distributed sensing as a framework in which to study distributed control.

[**Davis 76**] Davis, R., and J. King, "An Overview of Production Systems," in E. W. Elcock, and D. Michie (eds.), *Machine Intelligence 8*, Wiley, New York (1976), pp. 300–332. Broadly speaking, a production system is a set of pattern-action pairs (productions) and a database. A production system executes by repeatedly (1) finding a production whose pattern matches the database, and (2) executing the action of that production. The matching process on the pattern may bind some of the identifiers of the action, so one can write quantified productions. Production systems are a recurrent theme in A.I., appearing in many different guises. This paper discusses the theoretical nature of production systems and provides examples of systems that use them.

[**Davis 83**] Davis, R., and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving," *Artif. Intell.*, vol. 20, no. 1 (January 1983), pp. 63–109. Davis and Smith argue that negotiation is the appropriate metaphorical foundation of distributed problem solving and that Contract Nets is a good organization for such negotiation.

[**Erman 80**] Erman, L. D., F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Comput. Surv.*, vol. 12, no. 2 (June 1980), pp. 213–253. Hearsay-II is a speech understanding system, developed at Carnegie-Mellon University in the mid 1970s. The system is characterized by many "knowledge sources," each of which is an expert on some aspect that contributes to understanding the spoken sound. These sources communicate their conclusions about an input sound signal by writing messages on a common "blackboard."

[**Fennell 77**] Fennell, R. D., and V. R. Lesser, "Parallelism in AI Problem-Solving: A Case Study of Hearsay-II," *IEEE Trans. Comput.*, vol. C-26, no. 2 (February 1977), pp. 98–111. Fennell and Lesser analyzed the performance of a multiprocessor implementation of

Hearsay-II. They found that the system performance could improve by a factor of 4 to 6 by the use of multiprocessors.

[**Fox 79**] Fox, M. S., "Organization Structuring: Designing Large Complex Software," Technical Report CMU-CS-79-155, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (December 1979). Fox surveys organization and economic decision theory as it applies to organizing distributed programs and systems. In the final chapter of this report, he presents an organization design language, ODL.

[**Fox 81**] Fox, M. S., "An Organizational View of Distributed Systems," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-11, no. 1 (January 1981), pp. 70–80. Fox argues that management science has studied the principles of human (business) organizations and that these principles are applicable to organizing distributed problem-solving systems.

[**Galbraith 77**] Galbraith, J., *Organizational Design*, Addison-Wesley, Reading, Massachusetts (1977). Galbraith describes the general management concepts used in business organizations.

[**Hanson 78**] Hanson, A. R., and E. M. Riseman, "VISIONS: A Computer System for Interpreting Scenes," in A. Hanson, and E. Riseman (eds.), *Computer Vision Systems*, Academic Press, New York (1978), pp. 303–333. This paper describes a system that generates a high-level, semantic description of color images of natural scenes. The system architecture was strongly influenced by the design of Hearsay-II.

[**Hewitt 69**] Hewitt, C. E., "PLANNER: A Language for Manipulating Models and Proving Theorems in a Robot," *Proc. 1st Int. J. Conf. Artif. Intell.*, Washington, D.C. (August 1969), pp. 295–302. The problem-solving language Planner was Hewitt's dissertation. Knowledge in Planner was encapsulated in functional, pattern-invoked demons. Micro-Planner [Sussman 70] was an implementation of some of the ideas in Planner.

[**Hewitt 84**] Hewitt, C. E., and P. de Jong, "Message Passing Semantics for Conceptual Modeling," in M. L. Brodie, J. L. Mylopoulos, and J. W. Schmidt (eds.), *On Conceptual Modelling*, Springer-Verlag, New York (1984), pp. 147–164. Hewitt and de Jong present the theory of Open Systems.

[**Kent 81**] Kent, E., *The Brains of Men and Machines*, Byte/McGraw-Hill, Peterborough, New Hampshire (1981). In this book, Kent attempts to describe neural function and organization in terms that an electrical engineer can understand. The book is interesting not so much as an exact description of neurophysiology but as a sourcebook of ideas for the possible structure of intelligent systems.

[**Kornfeld 81**] Kornfeld, W. A., and C. E. Hewitt, "The Scientific Community Metaphor," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-11, no. 1 (January 1981), pp. 24–33. This paper projects the traditional paradigms of scientific discovery into an organization for a distributed problem solver.

[**Kornfeld 82**] Kornfeld, W. A., "Combinatorially Implosive Algorithms," *CACM*, vol. 25, no. 10 (October 1982), pp. 934–938. Kornfeld argues that parallel algorithms can form a "best-first" search strategy for search problems.

[**Lesser 78**] Lesser, V. R., and D. D. Corkill, "Cooperative Distributed Problem Solving: A New Approach for Structuring Distributed Systems," Technical Report 78-7, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts (May 1978). This report is an early presentation of the idea of cooperative distributed problem solving. It is the source of the distributed bus problem and the attacking-rovers game.

[**Lesser 80**] Lesser, V. R., and L. D. Erman, "Distributed Interpretation: A Model and Experiment," *IEEE Trans. Comput.*, vol. C-29, no. 12 (December 1980), pp. 1144–1163. Lesser and Erman describe the creation of a "distributed" version of the Hearsay-II speech understanding system.

[**Lesser 81**] Lesser, V. R., and D. D. Corkill, "Functionally Accurate, Cooperative Distributed Systems," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-11, no. 1 (January 1981), pp. 81–99. Lesser and Corkill assert that distributed systems need to treat uncertainty and errors as part of the network problem-solving process, much as some Artificial Intelligence systems

treat noisy input data and approximate knowledge in their problem solving. They discuss the paradigm of functionally accurate, cooperative systems in the context of distributed interpretation, distributed traffic control, and distributed planning.

[**Moore 79**] Moore, R. C., "Reasoning About Knowledge and Action," Ph.D. dissertation, M.I.T., Cambridge, Massachusetts (February 1979). Traditionally, reasoning about knowledge deals with determining what an individual could know. Several modal logics have been proposed for this task. Moore axiomatizes the possible-world semantics of modal logic in first-order logic. This leads to reasoning about the worlds that are compatible with an individual's knowledge.

[**Nii 78**] Nii, H. P., and E. A. Feigenbaum, "Rule-Based Understanding of Signals," in D. A. Waterman, and F. Hayes-Roth (eds.), *Pattern Directed Inference Systems*, Academic Press, New York (1978), pp. 483–501. SU/X is a system for interpreting large quantities of "continuous signals produced by objects" (sonar readings). SU/X uses the Hearsay-II architecture, principally the concepts of blackboard and multilevel representation of knowledge.

[**Smith 78**] Smith, R. G., and R. Davis, "Distributed Problem Solving: The Contract Net Approach," *Proc. 2d Natl. Conf. Canadian Soc. Comput. Stud. Intell.*, Toronto (July 1978), pp. 278–287. This paper is an overview of Contract Nets. It describes the distributed-sensing Contract Net.

[**Smith 81a**] Smith, R. G., *A Framework for Distributed Problem Solving*, UMI Research Press, Ann Arbor (1981). This dissertation is a general study of Contract Nets. It includes performance analyses of a simulated Contract Nets system and comparisons with other problem-solving formalisms.

[**Smith 81b**] Smith, R. G., and R. Davis, "Frameworks for Cooperation in Distributed Problem Solving," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-11, no. 1 (January 1981), pp. 61–70. Smith and Davis identify two forms of cooperation in a distributed problem-solving system: task sharing and result sharing. They discuss these two kinds of sharing with respect to Contract Nets.

[**Sussman 70**] Sussman, G. J., T. Winograd, and E. Charniak, "MICRO-PLANNER Reference Manual," Memo 203, Artificial Intelligence Laboratory, M.I.T., Cambridge, Massachusetts (1970). Micro-Planner is an implementation of some of the ideas in Hewitt's thesis [Hewitt 69]. Micro-Planner featured a database of "facts," pattern-directed invocation of demons that matched the facts in that database, and automatic backtracking. Micro-Planner had a period of popularity in the early 1970s. However, the automatic backtracking mechanism proved too cumbersome and the language fell into disuse.

[**Weyhrauch 80**] Weyhrauch, R. W., "Prolegomena to a Theory of Mechanized Formal Reasoning," *Artif. Intell.*, vol. 13, no. 1 (1980), pp. 133–170. Weyhrauch describes the knowledge representation system FOL. FOL encapsulates both collections of facts and rules for manipulating those facts in a single structure. One can both reason in this structure or reason about it.

[**Winograd 75**] Winograd, T., "Frame Representations and the Declarative-Procedural Controversy," in D. G. Bobrow, and A. Collins (eds.), *Representation and Understanding*, Academic Press, New York (1975), pp. 185–210. Prior to Micro-Planner, the mainstream of A.I. knowledge representations was declarative: the facts of the situation were described in a suitable logic and a general-purpose theorem prover sought to prove the desired goals. Micro-Planner represented knowledge in procedural form; knowing something was knowing what to do with it. This dichotomy lead to the "declarative-procedural controversy" as to whether the best method of knowledge representation used axioms or programs. A more modern view is that systems not only need to reason with their knowledge (procedural form) but also to reason about it (declarative form).